PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 17/30 — A1

(11) International Publication Number: WO 95/21416

(43) International Publication Date: 10 August 1995 (10.08.95)

(21) International Application Number: PCT/US95/01423

(22) International Filing Date: 3 February 1995 (03.02.95)

(30) Priority Data:
8/191,874 4 February 1994 (04.02.94) US

(71) Applicant: CADENCE DESIGN SYSTEMS, INC. [US/US]; 555 River Oaks Parkway, San Jose, CA 95134 (US).

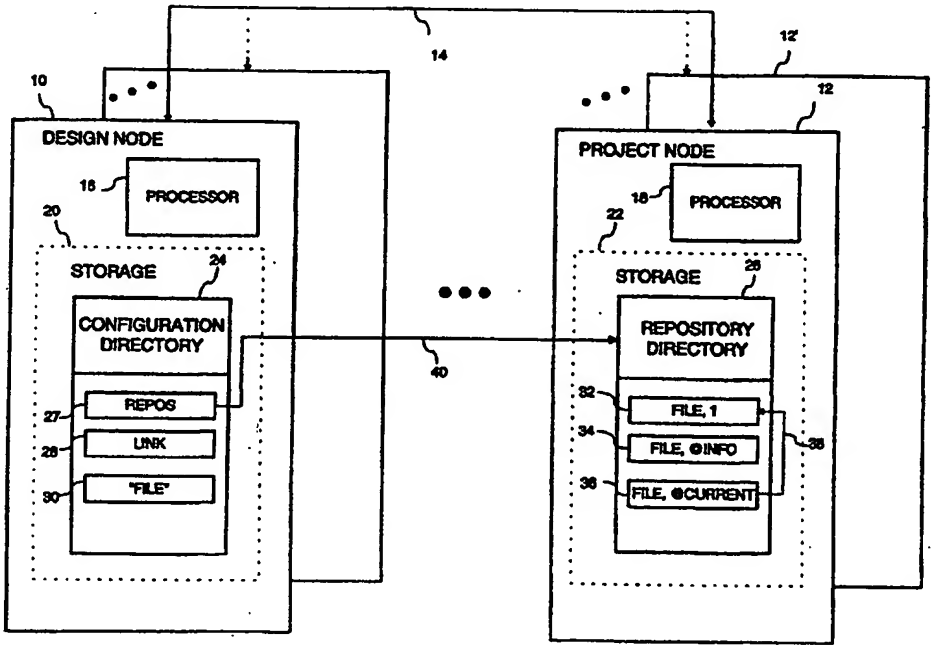
(72) Inventors: LOWE, Mitchell, Richard; 13425 Sycamore Drive, Morgan Hill, CA 95037 (US). EDWARDS, Russell, Paul; 1604 Mission Springs Circle, San Jose, CA 95051 (US).

(74) Agents: WELLER, Edward, B. et al.; Fenwick & West, Two Palo Alto Square, Suite 600, Palo Alto, CA 94306 (US).

(81) Designated States: CA, JP, KR, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: DISTRIBUTED FILE SYSTEM PROVIDING TRANSPARENT DATA MANAGEMENT



(57) Abstract

An electronic file management system enables virtual read access to files (30) in a configuration directory (24) by effectively accessing actual copies of such files (32, 34, 36) in a repository directory (26) using symbolic referencing between the directories (40). User-defined symbolic references specify whether static or dynamic linking is established.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	**GB**	United Kingdom	**MR**	Mauritania
AU	Australia	**GE**	Georgia	**MW**	Malawi
BB	Barbados	**GN**	Guinea	**NE**	Niger
BE	Belgium	**GR**	Greece	**NL**	Netherlands
BF	Burkina Faso	**HU**	Hungary	**NO**	Norway
BG	Bulgaria	**IE**	Ireland	**NZ**	New Zealand
BJ	Benin	**IT**	Italy	**PL**	Poland
BR	Brazil	**JP**	Japan	**PT**	Portugal
BY	Belarus	**KE**	Kenya	**RO**	Romania
CA	Canada	**KG**	Kyrgystan	**RU**	Russian Federation
CF	Central African Republic	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CG	Congo			**SE**	Sweden
CH	Switzerland	**KR**	Republic of Korea	**SI**	Slovenia
CI	Côte d'Ivoire	**KZ**	Kazakhstan	**SK**	Slovakia
CM	Cameroon	**LI**	Liechtenstein	**SN**	Senegal
CN	China	**LK**	Sri Lanka	**TD**	Chad
CS	Czechoslovakia	**LU**	Luxembourg	**TG**	Togo
CZ	Czech Republic	**LV**	Latvia	**TJ**	Tajikistan
DE	Germany	**MC**	Monaco	**TT**	Trinidad and Tobago
DK	Denmark	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**US**	United States of America
FI	Finland	**ML**	Mali	**UZ**	Uzbekistan
FR	France	**MN**	Mongolia	**VN**	Viet Nam
GA	Gabon				

DISTRIBUTED FILE SYSTEM PROVIDING TRANSPARENT DATA MANAGMENT

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates generally to the field of distributed processing and storage systems, particularly to methods for improving data management in such distributed systems.

2. Description of Background Art

Design of electronic systems through the use of computer-aided design (CAD) software has and continues to have a dramatic increase in complexity associated with exponential increases in the size and number of data files, number of people involved, and design tools from an increasingly varied set of design tool vendors. There is tremendous pressure to reduce design cycle times, while the complexity of the designs also increases exponentially. Management of the versions of the computer files representing these designs must now be automated given the current and increasing complexity (beyond the ability to cope via manual methods).

The standard approach to this problem is to create a database to represent all of this information, define and store a complex model of all the data files in that database, create a programming interface to access the information in the database, and copy files

out of the system when tools need to access the files.

The problems with this approach are increasing system complexity to the point that nobody can understand the system as a whole, performance problems as additional tools and users attempt to access the database(s), the inability to integrate all the design tools into the design management system, and the lack of a simple, well-defined method for using the system to achieve the desired result. Further, the ability to customize these systems is very low.

SUMMARY OF THE INVENTION

The invention resides in an electronic file management system having at least one processor and corresponding storage device wherein a configuration and repository directory are provided, such that a first symbolic reference in the configuration directory links to the repository directory, and a second symbolic reference in the configuration directory links through the first symbolic reference to a file in the repository directory. Thus, access of the file in the configuration directory effectively accesses the file in the repository directory.

Preferably, the invention defines a software-implemented method for utilizing the decentralized

facilities provided through distributed storage on a computer network made available by distributed file system constructs combined with a prescribed method for using the system to track and process changes made by design engineers, for example, in designs of electronic systems and to combine and integrate those changes into a coordinated set of files that define a consistent state of the electronic system for release to other organizations, including manufacturing groups, both in local and remote physical locations.

Through this invention, complex electronic systems can be designed by a larger number of engineers than otherwise possible without the system, requiring less training of the engineers and less diversion from actual design work, on a distributed network of computers, without undue hardship created by the failure from time to time of portions of the computer network or computers on the network, utilizing the latest and most advanced design tools without any special integration requirements. Designs developed by many engineers can be more easily and accurately assembled into complete and consistent systems. The system is easily customized to optimize the methodology used to assemble electronic systems.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a generalized block diagram of a system according to the present invention.

FIG. 2 is a simplified flow chart of a method according to the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

FIG. 1 shows a general block diagram of electronic file management system having one or more processor-based 16, 18 nodes, systems or engineering workstations 10, 12, which may be networked 14 using conventional techniques to other such nodes. Each node 10, 12 includes conventional storage devices 20, 22, such as semiconductor and/or disk-based digital memory units.

Preferably, node 10 serves as designer workstation, and node 12 serves as project manager workstation, in the context of a distributed engineering data system for managing electronic design, particularly managed object configurations or file sets in a given project. Thus, nodes 10, 12 may also include in storage 20, 22 an operating system, such as UNIX or VMS, and various computer-aided engineering and design (CAE/CAD) software tools, for example, for circuit schematic

capture, logic synthesis, digital or analog simulation, system layout, test vector generation, design documentation, database management, etc.

It is contemplated that nodes 10, 12 may be configured, nonetheless, in the context of other distributed software applications, particularly where complex projects give rise to the generation and management of a large number of files or directories located at multiple storage sites.

Further, it is contemplated that optional nodes 10', 12' having corresponding parts such as processors 16', 18', storages 20', 22', configuration and repository directories 24', 26', and so on, are distributed across and networked 14 to the system. Thus, as used herein, the apostrophe (') associated with a reference number represents corresponding parts for optional implementations, such as nodes 10', 12' which are functionally equivalent to primary parts corresponding in nodes 10, 12.

In accordance with the present invention, an electronic file management system or technique enables object or file access by an external software program or other user tool from one to another (or the same) directory, in a flexible and transparent manner.

In particular, using conventional operating system commands, a first or "configuration" storage portion or directory hierarchy 24 is provided 42, preferably in storage 20 of design node 10, and a second or "repository" storage portion or directory hierarchy 26 is provided 42, preferably in storage 22 of project node 12, as shown in the simplified flow chart of FIG. 2. Directories 24, 26 may be provided in the same node or may be a common directory. (See Appendix-A for simplified sample for directory creation.)

Optionally, a third "shadow" storage portion or directory 26' is similarly provided, including the same file information and directory structure included in repository directory 26. In this way, fault tolerance and data redundancy is provided in case repository directory 26 becomes corrupted or inaccessible.

Next, first symbolic reference or link file 40 is provided 44, within configuration directory 24, which defines a direct, virtual data coupling, preferably from repository link file 27 provided in configuration directory 24, to repository directory 26. Preferably, at any given moment, the repository link in configuration directory 24 is linked symbolically to only one repository directory 26.

As used herein, a symbolic reference or "link" is a file system special file type that points or refers

to another named file or object. Such links can span multi-processor or storage device file systems and point to files, directories or other links provided therein. The named file or object to which a link points is indicated by link contents, which include file system path name to the file or object to which such link points. Thus, when an operating system operation (e.g., read, write, close, open, etc.) is applied to a link, such operation effectively acts on the file to which the links points.

Preferably, symbolic reference 40 is provided in network or global name space or path name. A network path name is a file system path name that, when used on one computer or node, results in access of a file or object that resides on another computer or node coupled thereto. Additionally, a global path name is a network path name that can be used on each computer or node coupled to a network (e.g., local or wide area), which resolves to a common file on one of such computers.

Then, at least one managed object or data file 32, preferably identified initially as "file,1", is provided 46 or copied into repository directory 26 or sub directory thereunder. As described herein, a file set having more than one version of file 32 may be provided 46 therein. A "managed object" may be composed of many files 32, 34, 36 and optional additional files 32' contained in the repository directory 26. Files

that make up a managed object may have paths of the form: <data-file path>,<managed object extension>. The "managed object extension" is a unique string added to the names of the files composing the managed object 32, 34, 36 stored in the repository directory 26, in order to uniquely distinguish them from each other. (See Appendix-B for simplified sample of managed object creation.)

Preferably, file 32 is provided 46 in repository directory 26 initially by providing such file 32, as "file" 30, in configuration directory 24, possibly generated from external program or tool. Versions of file 30, each version having a unique, positive non-zero integer managed object extension and possibly containing different information, may be created using the check-in process, for instance, when user checks out files for editing and then returns edited file versions therein. At check-in, a corresponding link file 28 is created for the version created of file 30.

To provide each version of file 32 in repository directory 26, file 30 is copied from configuration directory 24 to repository directory 26; then, file 30 is deleted from configuration directory 24. In this way, link 28, having the same file name as deleted file 30, may be provided in configuration directory 24 to refer symbolically to copied file 32 in repository directory 26 through link 40, as described further

herein.

Optionally, associated text file 34 related to file 32 is provided, preferably identified as "file,@info", in the same directory or sub directory where file 32 is provided. Preferably, associated file 34 is provided for each managed object or file 32, including information thereof, such as version history, properties, status, user descriptions, check-out/in, etc. For example, file 34 is updated with version number reserved by a check-out operation or version number from which a new managed object is created by an initial checkin operation.

Preferably, file 32 is provided with a unique file name, which relates to a name assigned to corresponding file 28, 30 defined by user or tool. In this way, more than one version of file 32 may be stored in repository directory 26, and the system may detect when more than one file 32 is provided 46 with the same file name into repository directory or sub directory thereunder.

As described herein, check-in operation is the mechanism which saves current state of file 30 within repository directory 26. (See Appendix-C for simplified sample of check-in process.) In particular, this operation is done by copying contents of file 30 into a managed object version file 32 within repository

directory 26. After copying file 30 into version file 32, operating system access permissions of version file 32 are set not to allow additional modification of version file contents. This restriction is done to preserve contents as they were when check-in operation was performed.

Additionally, the copy of data file within configuration directory 24 is deleted and replaced by a symbolic reference, known as the managed object reference 28 to managed object version into which file 30 was copied 32. The managed object version into which file 30 is copied must have been created by checkout-for-write operation, or never have been checked in before, as described herein. The same user performs both checkout-for-write and check-in operations because version to copy data file into is determined by which version file was checked out by user performing check-in. Preferably, this determination is done by examining contents of managed object's "co.<user-name>" link.

If there is no version file checked out by the user performing the check-in operation, such operation will fail, and an error is reported. After successfully copying file 30 into version file 32, the managed object's "co.<user-name>" link is removed from repository directory 26. Whenever file 30 is checked in to managed object version file 32 by check-in operation, contents of managed object's "@current" link 36 is

modified to point to version file 32. This modification causes dynamic Managed Object References to the managed object to refer automatically to newly checked-in version.

Check-in operation requires three pieces of data on which to operate: (1) configuration directory path 24, (2) file 30 path within configuration directory 24, and (3) binding type specification. Configuration directory path identifies configuration directory 24 and repository directory 26, via configuration directory's repository link 27, which check-in operation is to use; data file path identifies which data file is to be operated on; and binding type specification indicates what type of managed object reference to create.

The act of performing check-in operation on file 30 is termed "checking the data file in" and after completion, such data file is said to have been "checked in." Check-in operation does not operate on managed object references (such as 28). Performing check-in operation on data file twice in a row is not permitted because the first check-in operation replaces data file with managed object reference. Note that checkout-for-write operation may remove managed object reference, thereby replacing such reference with writable copy of one of managed object's version files.

Further, second symbolic reference or link file 28 is provided 48, which defines a direct, virtual data coupling through, or which includes effectively, symbolic reference 40 to file 32. This second symbolic reference is termed a "Managed Object Reference." Preferably, symbolic reference 28 is provided in local name space in configuration directory 24 or sub directory therein, according to user-defined link specification or binding type, which indicates type of symbolic reference 28.

Two types of reference to managed object file 32 can be defined by link 28. One type of reference is termed "static." A static reference is one in which link 28 includes path to link 40 and then the path to file 32 directly within repository directory 26.

Another type of reference is termed "dynamic." To provide a dynamic reference, third symbolic reference or link file 36 is created in repository directory 26 which includes path 38 to file 32. Preferably, link 36 is identified as "file,@current" and serves to determine which version file is referenced by dynamic managed object reference. Further, to provide such dynamic reference, link 28 includes path to link 40 and path to link 36 within repository directory 26. Thus, when read operation 50 on link 28 is performed (e.g., by external program or tool,) the effective result of read operation

50 is the same, such that content of file 32 is read.

Hence, by changing the file to which link 28 references, the file referenced may be redefined. For example, in a static case, link 28 must be changed to refer to a different file, other than file 32, such that link 28 points to another version of file 32, preferably provided in repository directory 26 or sub directory therein.

Additionally, in a dynamic case, either link 28 or link 36 may be changed to achieve the same result (i.e., accessing a particular version of file 32). For example, link 36 may be caused to refer to version-X of file 32, without changing content of link 28, but effectively changing the file to which link 28 points. Thus, dynamic referencing allows link 28 to be fixed once and then remain unchanged, while permitting the file to which accessing is performed to be changed by changing link 36. Note that during the file check-in process, content of link 36 may be changed to point to most recently-created file version.

Preferably, managed object references contain three path name components: (1) reference to repository link 27, (2) directory portion of path to file 32, and (3) file name of one of managed object files for data file. In particular, the repository link reference portion is a "relative path" from the directory that the managed object reference is in to the repository link

for configuration directory 24. Because of how symbolic links function, this relative path allows contents of the managed object reference to determine the particular repository directory 26 to which the managed object reference refers. When the managed object reference and repository link are in the same directory, this portion merely contains the name of repository link.

Moreover, the managed object file name component that is selected depends on type of managed object reference. Hence, in the case of static managed object references, the file name selected is the managed object version corresponding to the version specified in the configuration reference. Preferably, such static references always refer to the same managed object version.

In the case of dynamic managed object references, the file name selected is the managed object's "@current" symbolic link. As described further herein, such dynamic links enable the managed object version selected to be controlled by contents of managed object's "@current" link.

Optionally, an other storage portion or configuration directory 24' is provided, on any node 10' on system network 14, which may include information different from that included in primary configuration directory 24. In this implementation, third symbolic

reference or link 40' is provided, which defines a direct, virtual data coupling from the Repository Link 27' in the other configuration directory 24', or sub directory thereunder, to repository directory 26, or sub directory thereunder. In this way, fourth symbolic reference 28' provided in corresponding storage 20' of other node 10' similarly links file 32 or other version of file 32' or version of different file 32, 34 provided in repository directory 26 through, or which includes effectively, symbolic reference 40'.

Additionally, an other repository storage portion or directory 26' may be provided, on any node 12' on system network 14, which may include information different from that included in primary repository directory 26. In this implementation, fifth symbolic reference or link 40' is provided, which defines a direct, virtual data coupling from the Repository Link 27' in the configuration directory 24, or sub directory thereunder, to other repository directory 26', or sub directory thereunder. In this way, sixth symbolic reference 28' provided in storage 20' of design node 10' similarly links an other object or file 32' provided in other repository directory 26' through, or which includes effectively, symbolic reference 40'.

After symbolic reference 28 is provided 48, file 32 may be accessed through such symbolic reference 28 to perform read operation 50. Symbolic Reference 28 is

created preferably through checkout-for-read process which provides a reference to the read-only copy of file 32. Alternately, any tool or user program may now access file 32 through Symbolic Reference 28 withouth knowing its version or true location. (See Appendix-D for simplified sample of check-out process for read.)

Preferably, checkout-for-read operation is used to create managed object reference in configuration directory 24. As described further herein, such operation requires three pieces of information on which to operate: (1) configuration directory path, (2) data file path within configuration directory, and (3) binding type specification.

In particular, configuration directory path identifies configuration directory 24 and repository directory 26 via configuration directory's repository link 27, on which checkout-for-read operation will operate. Additionally, data-file path identifies which managed object the managed object reference is for, and binding type specification indicates what type of managed object reference to create.

Optionally, version specification is provided for use when static binding type is specified. If dynamic binding type is specified and version specification is provided, version specification is ignored because dynamic managed object reference points

to "@current" link and not a specific version.

No modification of a managed object's files is performed by checkout-for-read operation. This restriction is done explicitly to allow users to perform checkout-for-read operation on managed objects which they may not be able modify, because of operating system access restrictions. The act of performing checkout-for-read operation on a data file may be termed "checking the data file out for reading" and after completion, data file may be said to have been "checked out for reading."

Checkout-for-read operation does not operate if there is a writable copy of data file in specified configuration directory 24. This restriction is done to prevent accidental loss of data by unintentional use of checkout-for-read operation. If this situation is encountered, such checkout operation will fail, and error is reported. In any other situation, checkout-for-read operation will remove any copy or managed object reference for data-file in configuration directory 24 and replace it with specified managed object reference.

Alternately, to perform write operation 52, file 32 is copied, preferably using a checkout-for-write process to generate a modifiable or writable file copy in configuration directory 24 or a sub-directory

thereof. Such checkout process is used to create a modifiable (i.e., "checked-out") managed object version file for the user performing write operation 52. (See Appendix-E for simplified sample of check-out process for write.)

In particular, an existing managed object version, which cannot be modified ordinarily, is copied to create another checked-out version, which is modifiable. Also, a managed object check-out link, which records which version was checked-out, may be created by user performing such check-out operation. This link is usable during check-in to determine into which managed object version file the writable file is to be copied.

If the user performing a check-out for write operation already has a checked-out version of the managed object, then such operation fails, and error is reported. Preferably, each user may check out one version at a time of a managed object.

The checkout-for-write operation requires three pieces of information on which to operate: configuration directory 24 path, data file path within configuration directory 24, and version selection option. Configuration directory path identifies configuration directory 24 and repository directory 26, for example, via configuration directory's repository link 27, 40, on

which such operation operates. Data file path identifies managed object on which to operate, and version selection option identifies how version to copy is selected.

There are three different version selection options which may be specified: (1) Current: this option indicates that version to be copied should be the version pointed at by the managed object "@current" link. (2) Reference: this option indicates that version to be copied should be the version pointed at by the managed object reference for data file in configured directory 24. (3) Version <version_number>: this option indicates that a version <version_number> is to be copied. If version <version_number> does not exist, then checkout-for-write operation fails and reports error.

The checked out version is a modifiable copy of specified version 32, which is stored in configuration directory 24 as specified to checkout-for-write operation. Preferably, managed object's "@info" file 34 is examined to find the largest version number used to date for such managed object. The new version is assigned a version number which is one larger than that, and an entry for the newly created version is added to the "@info" file.

The new "@info" file entry records version number of the newly created version and also records

which version number from which it was copied. These entries represent the version derivation history of managed object (i.e., which versions were created from which others).

A managed object checkout symbolic reference or link file is created in repository directory 26 for the user performing checkout-for-write operation. This link is named "co.<user-name>", where user-name is replaced by the name of users account on node running checkout-for-write operation. Contents of such link is the name of the newly created managed object version file.

Appendix A

SIMPLIFIED SAMPLE OF CREATING A CONFIGURATION DIRECTORY

```
ConfigDirPath = [Get Path to Configuration Directory]
ReposDirPath  = [Get Path to Repository Directory]
RepositoryLinkName = ".raw_data"
RepositoryLinkPath = ConfigDirPath/RepositoryLinkName

if ( ConfigDirPath does not exist) {
    create directory ConfigDirPath
    if (error while creating) {
    print error
    quit
    }
}
if ( RepositoryLinkPath exists) {
    print "Error - ConfigDirPath is already a
Configuration Directory"
    quit
}
create symbolic link RepositoryLinkPath with contents
ReposDirPath

if (error while creating link) {
    print error
    quit
}

Done
```

Appendix B

SIMPLIFIED SAMPLE OF CREATING A MANAGED OBJECT

```
ConfigDirPath      = [Get ConfigDirPath]
DataFilePath       = [Get data file-path]
ConfigDataFilePath = ConfigDirPath/DataFilePath
RepositoryLinkName = ".raw_data"
RepositoryLinkPath = ConfigDirPath/RepositoryLinkName
DataFileName       = filename portion of DataFilePath

if (ConfigDirPath does not exist) {
   print "Error - Configuration Directory does not exist"
   quit
}

if (ConfigDataFilePath does not exist) {
   print "Error - ConfigDataFilePath does not exist"
   quit
}

if (RepositoryLinkPath does not eist) {
   print "Error - ..."
   quit
}

RepositoryDirectoryPath = contents of RepositoryLinkPath symbolic link
MOPath = RepositoryDirectoryPath/DataFilePath
InfoPath = MOPath + ",@info"
```

```
if (InfoPath exists) {
    print "Error - MO already exists for DataFilePath"
    quit
}

if (MOPath exists and is a directory) {
    print "Error - DataFilePath is already Checked in as a Directory"
    quit
}

VersionPath = MOPath + ",1"

for all directories in MOPath {
    if (directory does not exist) {
        create directory
        if (error while creating) {
            print error
            quit
        }
    }
}
InfoRecord = "1 0"
Open InfoPath for Write
Write InfoRecord to InfoPath
Close InfoPath

Copy ConfigDataFilePath to VersionPath
if (error) {
    print "Error = Unable to copy file"
```

```
    quit
}

Change Access permission of VersionPath to non-modifiable

CurrentPath = MOPath + ",@current"
CurrentContents = DataFileName + ",1"

Create Symbolic link CurrentPath with Contents CurrentContents

RepositoryLinkRelativePath = ""
for each directory in DataFilePath {
    RepositoryLinkRelativePath = RepositoryLinkRelativePath + "../"
}
RepositoryLinkRelativePath = RepositoryLinkRelativePath +
RepositoryLinkName

if (BlindingType = STATIC) {
    MORContents = RepositoryLinkRelativePath + DataFilePath + ",1"
}
if (BlindingType = DYNAMIC) {
    MORContents = RepositoryLinkRelativePath + DataFilePath +
",@current"
}
Delete ConfigDataFilePath
Create symbolic link ConfigDataFilePath with contents MORContents
Done
```

Appendix C

SIMPLIFIED SAMPLE OF CHECK-IN PROCESS

```
ConfigDirPath       = [Get ConfigDirPath]
DataFilePath        = [Get data-file-path]
BindingType     = [Get BindingType value]
ConfigDataFilePath = ConfigDirPath/DataFilePath
RepositoryLinkName = ".raw_data"
RepositoryLinkPath = ConfigDirPath/RepositoryLinkName
DataFileName        = filename portion of DataFilePath

if (ConfigDirPath does not exist) {
   print "Error - Configuration Directory does not
exist"
   quit
}

if (ConfigDataFilePath does not exist) {
   print "Error - ConfigDataFilePath does not exist"
   quit
}

if (RepositoryLinkPath does not exist) {
   print "Error - ..."
   quit
}


RepositoryDirectoryPath = contents of
```

```
RepositoryLinkPath symbolic link
MOPath = RepositoryDirectoryPath/DataFilePath
InfoPath = MOPath + ",@info"
CheckoutPath = MOPath + ",@co." + <user-account-name>

if (InfoPath does not exist ) {
   print "Error - MO does not exist for DataFilePath"
   quit
}

if (CheckoutPath does not exist) {
   print "Error - user <user> does not have
DatafilePath checked out"
   quit
}

if (ConfDataFilePath does not exist) {
   print "Error - no file to checkin"
   quit
}
if (ConfigDataFilePath is a directory) {
   print "Error - cannot checkin a directory"
   quit
}

CheckoutPathContents = read contents of CheckoutPath
CheckedOutVersion = [parse version number from
CheckoutPathContents]
VersionPath = MOPath + "," + CheckedOutVersion
```

```
Copy ConfigDataFilePath to VersionPath
Set OS access permissions of VersionPath to non-
modifiable

CurrentPath = MOPath + ",@current"
CurrentContents = DataFileName + "," +
CheckedOutVersion
delete CurrentPath
create symbolic link CurrentPath with contents
CurrentContents

RepositoryLinkRelativePath = ""
for each directory in DataFilePath {
    RepositoryLinkRelativePath =
RepositoryLinkRelativePath + "../"
}

RepositoryLinkRelativePath = RepositoryLinkRelativePath
+ RepositoryLinkName
if (BindingType = STATIC) {
    MORContents = RepositoryLinkRelativePath +
DataFilePath + "," + CheckedOutVersion
}

if (BindingType = DYNAMIC) {
    MORContents = RepositoryLinkRelativePath +
DataFilePath + ",@current"
}
```

```
Delete ConfigDataFilePath
Create symbolic link ConfigDataFilePath with contents
MORContents

delete CheckoutPath link

Done
```

Appendix D

SIMPLIFIED SAMPLE OF CHECK-OUT FOR READ OPERATION

```
ConfigDirPath      = [Get ConfigDirPath]
DataFilePath       = [Get data-file-path]
BindingType        = [Get BindingType]
ConfigDataFilePath = ConfigDirPath/DataFilePath
RepositoryLinkName = ".raw_data"
RepositoryLinkPath = ConfigDirPath/RepositoryLinkName
DataFileName       = filename portion of DataFilePath

if (ConfigDirPath does not exist) {
   print "Error - Configuration Directory does not
exist"
   quit
}

if (ConfigDataFilePath is a writable file) {
   print "Error - ConfigDataFilePath is writable, this
operation will
         not overwrite it"
   quit
}

if (RepositoryLinkPath does not exist) {
   print "Error - ..."
   quit
}
```

```
RepositoryDirectoryPath = contents of
RepositoryLinkPath symbolic link
MOPath = RepositoryDirectoryPath/DataFilePath
InfoPath = MOPath + ",@info"

if (InfoPath does not exists) {
   print "Error - MO for DataFilePath does not exist"
   quit
}

if (MOPath exists and is a directory) {
   print "Error - DataFilePath is Checked in as a
Directory"
   quit
}

RepositoryLinkRelativePath = ""
for each directory in DataFilePath {
    RepositoryLinkRelativePath =
RepositoryLinkRelativePath + "../"
}

RepositoryLinkRelativePath = RepositoryLinkRelativePath
+ RepositoryLinkName

if (BindingType = STATIC) {

   VersionNumber = [Get Version Number]
   if (no Version Number provided) {
```

```
        CurrentPath = MOPath + ",@current"
        CurrentContents = read contents of CurrentPath
        VersionNumber = Parse Managed Object Extension
from CurrentContents
    }

    MORContents = RepositoryLinkRelativePath +
DataFilePath + VersionNumber

}

if (BindingType = DYNAMIC) {
    MORContents = RepositoryLinkRelativePath +
DataFilePath + ",@current"
}

delete ConfigDataFilePath
create symbolic link ConfigDataFilePath with contents
MORContents
```

Appendix E

SIMPLIFIED SAMPLE OF CHECK-OUT FOR WRITE OPERATION

```
ConfigDirPath          =          [Get ConfigDirpath]
DataFilePath           =          [Get data-file-path]
VersionSpec            =          [Get Version Specification]
ConfigDataFilePath     =          ConfigDirPath/DataFilePath
RepositoryLinkName     =          ".raw_data"
RepositoryLinkPath     =          ConfigDirPath/RepositoryLinkName
DataFileName           =          filename portion of DataFilePath
if (ConfigDirPath does not exist)   {
   print "Error  -  Configuration Directory does not exist"
   quit
}
if (ConfigDataFilePath exists and is writable)  {
   print "Error  -  ConfigDataFilePath is writable, this operation will
       not overwrite it"
   quit
}
if (RepositoryLinkPath does not exist)  {
   print "Error  -  . . ."
   quit
}
RepositoryDirectoryPath  =  contents of RepositoryLinkPath symbolic
   link
MOPath  =  RepositoryDirectoryPath/DataFilePath
InfoPath  =  MOPath  +  ",@info"
if (InfoPath does not exists)  {
   print "Error  -  MO for DataFilePath does not
       exist"
```

```
    quit
  }
  if (MOPath exists and is a directory){
  print "Error  -  DataFilePath is Checked in as a Directory"
    quit
  }
  if (VersionSpec  =  Reference)  {
    CurrentPath  =  MOPath  +  ",@current"
    CurrentContents  =  read contents of
        CurrentPath
  VersionNumber  =  parse version number from CurrentContents
  }
  if (VersionSpec  =  Current) {
    if (ConfigDataFilePath is not a symbolic link)  {
        print "Error  -  ConfigDataFilePath is not a Managed Object
        Reference"
        quit
    }
    MORContents  =  read contents of ConfigDataFilePath
    MORExten  =  parse MOR Contents for the Managed Object Extension
    if (error finding Managed Object Extension)  {
        print "Error  -  ConfigDataFilePath is not a Managed Object
        Reference"
        quit
    }
    if (MORExten is "@current")     {
        CurrentPath  =  MOPath  +  ",@current"
        CurrentContents  =  read contents of CurrentPath
        VersionNumber  =  parse version number from CurrentContents
```

```
    } else {
        /* MORExten must be a version number */
        VersionNumber = MORExten
    }
}
if (VersionSpec = Version) {
    VersionNumber = [Get Version Number]
}
CheckoutLinkPath = MOPath + ",@co." + user account name
if (CheckoutLinkPath exists) {
    print "Error - user <account name> already has DataFilePath
        checked out"
    quit
}
if (don't have OS access permission to modify InfoPath file OR
    don't have OS access permission to create file in InfoPath
        directory) {

    print "Error - lack sufficient permission to checkout
        DataFilePath"
    quit
}
VersionPath = MOPath = "," = VersionNumber
if (VersionPath does not exist) {
    print "Error"
    quit
}
LargestVersion = 0
Open InfoPath file for reading
```

```
for each Record in InfoPath   {
   Parse record into RecordVersionNumber and RecordDerivedFromVersion
   if (RecordVersionNumber  >  LargestVersion)
       LargestVersion  =  RecordVersionNumber
}
NewVersionNumber  =  LargestVersionNumber  +  1
Construct NewInfoRecord  =  NewVersionNumber  +  "  "  +  VersionNumber
Write NewInfoRecord to InfoPath
for each directory in ConfigDataFilePath   {
   if (directory does not exist)
       create directory
}
NewVersionPath  =  MOPath  +  ","  +  NewVersionNumber
CheckoutLinkContents  =  DataFileName  +  ","  +  NewVersionNumber
if (ConfigDataFilePath exists)   {
   delete ConfigDataFilePath
}
copy file VersionPath to ConfigDataFilePath
set OS permissions of ConfigDataFilePath to modifiable
create symbolic link CheckoutLinkPath with contents
CheckoutLinkContents
```

CLAIMS

We claim:

1. A method for electronic file management comprising the steps of:

providing a first and second directory;

providing a first link in the first directory to the second directory;

providing a first file in the second directory; and

providing a second link in the first directory or a sub-directory thereunder through the first link to the first file.

2. The method of Claim 1 wherein:

the first and second directory comprise a common directory.

3. The method of Claim 1 further comprising the step of:

providing a third directory that includes the same information as included in the second directory.

4. The method of Claim 1 wherein:

the first and second links comprise symbolic references.

5. The method of Claim 1 further comprising the step of:

providing a fourth directory that includes

different information than included in the second directory;

providing a third link in the first directory to the fourth directory;

providing a second file in the fourth directory; and

providing a fourth link through the third link to the second file.

6. The method of Claim 1 wherein:

the first file is provided in a sub directory under the second directory.

7. The method of Claim 1 wherein:

the second link is provided from a sub directory under the first directory.

8. The method of Claim 1 further comprising the step of:

providing in the second directory a third file associated with the first file.

9. The method of Claim 8 wherein:

the third file comprises version information about the first file.

10. The method of Claim 1 wherein:

the first file is provided with a first name, such that providing another file with the same name is

detectable.

11. The method of Claim 1 wherein:

a plurality of versions of the first file is provided in the second directory.

12. The method of Claim 1 wherein:

the second link is provided according to a user-defined link specification.

13. The method of Claim 12 wherein:

the link specification indicates whether the second link is static or dynamic.

14. The method of Claim 12 further comprising the step of:

providing in the second directory a third link which refers to the first file.

15. The method of Claim 14 wherein:

the second link to the first file is provided also through the third link.

16. The method of Claim 14 wherein:

the second link refers to the third link to refer indirectly to the first file.

17. The method of Claim 14 wherein:

the second link is pointable either to the first

file in static mode or to the third link in dynamic mode.

18. The method of Claim 14 wherein:

the third link is modifiable to refer to an other file.

19. The method of Claim 1 further comprising the steps of:

providing a fifth directory;

providing a fifth link in the fifth directory to the second directory; and

providing a sixth link in the firth directory through the fifth link to the first file in the second directory.

20. The method of Claim 1 further comprising the step of:

accessing the first file through the second link to perform a read operation.

21. The method of Claim 1 further comprising the steps of:

deleting the second link; and

copying the first file to the first directory to perform a write operation.

22. The method of Claim 1 wherein:

the first link is provided in a network name

space.

23. The method of Claim 1 wherein the step of providing the first file in the second directory comprises the following steps:

providing the first file in the first directory;

copying the first file from the first directory to the second directory;

deleting the first file in the first directory; and

providing the second link in the first directory to the copied first file in the second directory through the first link.

24. The method of Claim 1 wherein:

the second link is provided a link name which is identical to a file name provided to the first file.

25. The method of Claim 1 wherein:

the second link is provided with a link name which is identical to a file name provided to a copied file provided in the first directory, wherein the copied file is identical to the first file.

26. A method for object management comprising the steps of:

providing a configuration directory and a repository directory;

providing a first symbolic reference in the

configuration directory to the repository directory;

providing an object in the repository directory; and

providing a second symbolic reference through the first symbolic reference to the object, wherein the second symbolic reference is provided according to a user-defined specification which indicates whether the second symbolic reference is static or dynamic.

27. A method for electronic file management comprising the steps of:

providing a configuration directory and a repository directory;

providing a first symbolic reference in the configuration directory to the repository directory;

providing a file in the configuration directory;

copying the file from the configuration directory to the repository directory;

deleting the file in configuration directory; and

providing a second symbolic reference through the first symbolic reference to the file copy in the repository directory.

28. The method of Claim 27 wherein:

the second symbolic reference is provided according to a user-defined specification which indicates whether the second symbolic reference is static or dynamic.

29. An electronic file management system comprising:

at least one processor unit, each processor unit having a corresponding storage device;

a configuration directory in one of the storage units;

a repository directory in one of the storage units;

a first symbolic reference in the configuration directory to the repository directory;

a file in the repository directory; and

a second symbolic reference for linking through the first symbolic reference to the file;

wherein an access to the file in the configuration directory effectively accesses the file in the repository directory.

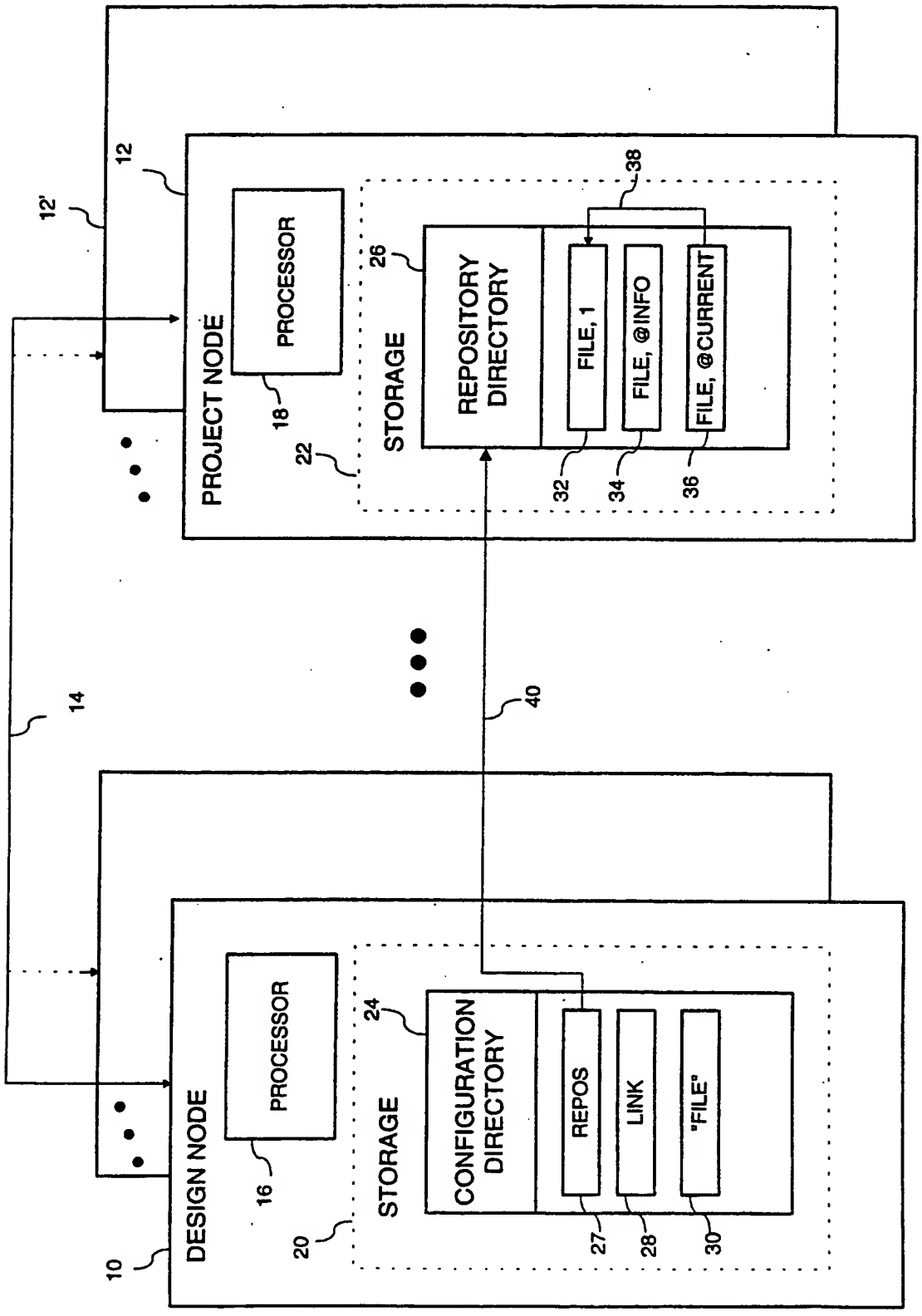

12'
12
PROJECT NODE
PROCESSOR
18
22
STORAGE
26
REPOSITORY DIRECTORY
38
FILE, 1
FILE, @INFO
FILE, @CURRENT
32
34
36
14
40
DESIGN NODE
10
PROCESSOR
16
20
STORAGE
24
CONFIGURATION DIRECTORY
REPOS
LINK
"FILE"
27
28
30

FIGURE 1

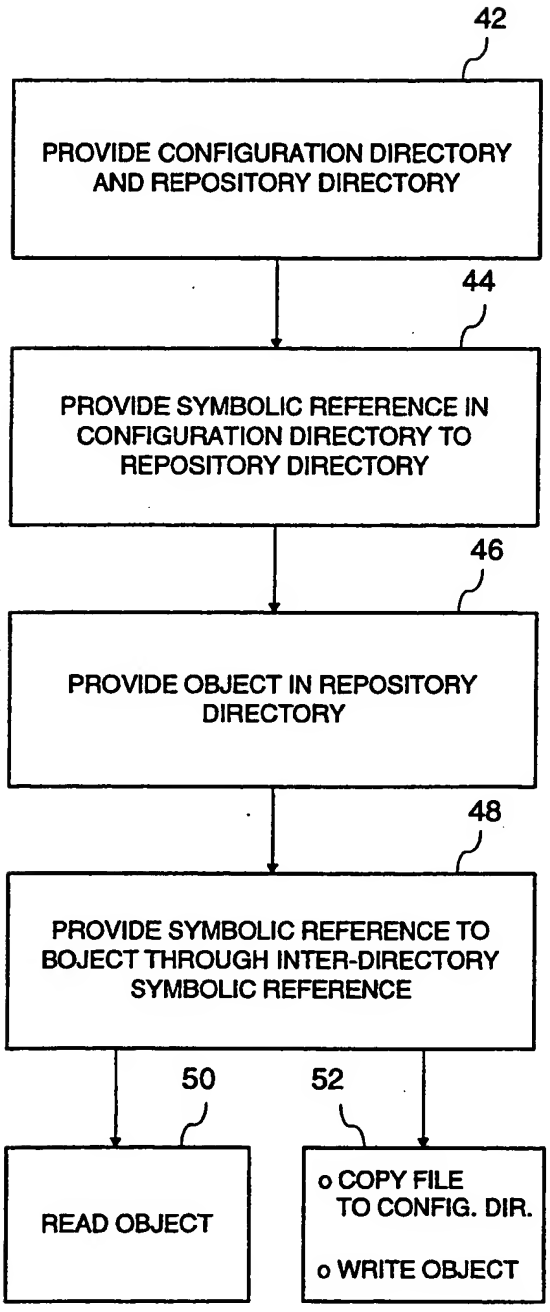

42
PROVIDE CONFIGURATION DIRECTORY AND REPOSITORY DIRECTORY
44
PROVIDE SYMBOLIC REFERENCE IN CONFIGURATION DIRECTORY TO REPOSITORY DIRECTORY
46
PROVIDE OBJECT IN REPOSITORY DIRECTORY
48
PROVIDE SYMBOLIC REFERENCE TO BOJECT THROUGH INTER-DIRECTORY SYMBOLIC REFERENCE
50
52
READ OBJECT
o COPY FILE TO CONFIG. DIR.
o WRITE OBJECT

FIGURE 2

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US95/01423

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 17/30
US CL : 395/600
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 395/600, 650, 700, 425

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X Y	MICROSOFT PRESS, 1988, R. DUNCAN, "THE MS-DOS ENCYCLOPEDIA", PAGES 279-281.	1-7,10,12,14-16,18-21,23 2,8,9,11,13,17,22,24-29
A,P	US, A, 5,355,497 (COHEN-LEVY) 11 OCTOBER 1994	1-29

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be part of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search	Date of mailing of the international search report
22 MARCH 1995	25 JUL 1995
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer ALLEN MACDONALD Telephone No. (703) 305-9646

Form PCT/ISA/210 (second sheet)(July 1992)*